1995-09

Design and specification of an object-oriented data definition language

Badgett, Robert B.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND SPECIFICATION OF AN OBJECT-ORIENTED DATA DEFINITION LANGUAGE

by

Robert B. Badgett

September 1995

Thesis Advisor: David K. Hsiao
Co-Advisor: C. Thomas Wu

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1995	Master's Thesis

4. TITLE AND SUBTITLE
Design and Specification of an Object-Oriented Data Definition Language

5. FUNDING NUMBERS

6. AUTHOR(S)
Robert B. Badgett

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The problem addressed by this thesis is the inability of traditional data models to efficiently support the new database applications of today, such as Computer-Aided Design and multimedia. Traditional data models were designed for specific business type applications, i.e., record keeping (relational) and product assembly (hierarchical). Because of this, their permitted data types, structures, and query languages are specific and therefore limited. New applications require more complex and varied data structures and data types. The flat representation of data by traditional data models results in complex objects being scattered over many relations losing the correspondence between the user's view and database representation.

The approach taken was to develope a new object-oriented data model (O-ODM). The object-oriented approach permits both the structure of complex objects and their operations to be specified by the designer, providing a flexibility not available in traditional data models. As a result an object may be modelled closer to the user's view, permitting the application programmer to easily capture its complexity.

The result of this thesis is the specification of an object-oriented data definition language (O-ODDL) for the O-ODM. The O-ODDL incorporates the features of a unique object, object classes, inheritance, the covering, and encapsulation. The covering, unique to this O-ODM, is important in that it maps an object in one class to a subset of objects in another, providing the ability to manipulate an object as either a singleton or set. This O-ODM and its O-ODDL provide the constructs necessary to represent the new database applications of today.

14. SUBJECT TERMS
Object-Oriented Data Model
Object-Oriented Data Definition Language
Object-Oriented Database

15. NUMBER OF PAGES
46

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

DESIGN AND SPECIFICATION OF AN OBJECT-ORIENTED DATA DEFINITION LANGUAGE

Robert B. Badgett
Commander, United States Navy
B.N.E., Georgia Institute of Technology, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1995

Author: [redacted]

Robert B. Badgett

Approved by: [redacted]

David K. Hsiao, Thesis Advisor

[redacted]

C. Thomas Wu, Co-Advisor

[redacted]

Ted Lewis, Chairman
Department of Computer Science

ABSTRACT

The problem addressed by this thesis is the inability of traditional data models to efficiently support the new database applications of today, such as Computer-Aided Design and multimedia. Traditional data models were designed for specific business type applications, i.e., record keeping (relational) and product assembly (hierarchical). Because of this, their permitted data types, structures, and query languages are specific and therefore limited. New applications require more complex and varied data structures and data types. The flat representation of data by traditional data models results in complex objects being scattered over many relations losing the correspondence between the user's view and database representation.

The approach taken was to develope a new object-oriented data model (O-ODM). The object-oriented approach permits both the structure of complex objects and their operations to be specified by the designer, providing a flexibility not available in traditional data models. As a result an object may be modelled closer to the user's view, permitting the application programmer to easily capture its complexity.

The result of this thesis is the specification of an object-oriented data definition language (O-ODDL) for the O-ODM. The O-ODDL incorporates the features of a unique object, object classes, inheritance, the covering, and encapsulation. The covering, unique to this O-ODM, is important in that it maps an object in one class to a subset of objects in another, providing the ability to manipulate an object as either a singleton or set. This O-ODM and its O-ODDL provide the constructs necessary to represent the new database applications of today.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank Dr. David K. Hsiao and Dr. C. Thomas Wu for their time and advice during the research and writing of this thesis. Without their knowledge and expertise this thesis would not have been possible.

I would also like to thank the support staff at the Naval Postgraduate School for their help. Particularly Sue Whalen for her help with the MDBS software.

Finally, I would like to thank my wife, Beth, and sons, Matthew and Scott, for their patience.

I. INTRODUCTION

The objective of my thesis is to define the specification of a Data Definition Language (DDL) for a new Object-Oriented Data Model (O-ODM). With respect to the O-ODM, I propose two questions which must be answered before any design of the DDL. The first question is as follows: Why is a new data model required and not the modification of an existing data model? The second question then becomes: Why is the object-oriented model chosen as the new in lieu of the modification of an existing model? In this introduction, I address both of these questions and demonstrate the need for a new data model, i.e., the O-ODM.

In answering the first question one only has to look at the applications and requirements placed on databases over the last decade. Initially databases and their corresponding data models were implemented mainly for business type applications. I will refer to these as traditional data models. Traditional models include the relational, network, hierarchical, and functional models. Today this is no longer true, there are many new applications which are significantly different than the previous traditional or business applications. These applications include areas such as Computer-Aided Design (CAD), Computer-Aided Manufacturing (CAM), Computer-Aided Software Engineering (CASE), Artificial Intelligence (AI), and multimedia. These newer applications are complex (a discussion of which is contained in the next paragraph) when compared to traditional applications. Their complexity requires a new and more powerful data model to characterize their data and to create their databases.

The complexity refers to the various data types in a data aggregate and various relationships among aggregated data in order to provide powerful data abstractions.With these abstractions the user can view the complex data readily in the context of the new application [1,2].

On the other hand, a traditional database model is designed for a specific type of application, which is referred to as *application specifity* [1]. For example, the relational data model is designed to model records, the hierarchial data model for product assembly, the network data model for inventory controls, and the functional data model, for infer-

ences. Here four distinct applications require four distinct data models to represent their data, i.e., a result of application specifity. These traditional data models do not adequately support new applications. The complexity of the new applications makes it impossible for a traditional data model to organize related data for the efficient access, retrieve and/or update of data. Therefore, for new applications, a new model is needed.

Now that I have established a new data model is required I answer the second question: Why the object-oriented data model? New programming languages and data structures have been created to support the advanced computing and software requirements. The object-oriented paradigm is rooted in the earliest object oriented programming language, Smalltalk. It was not until the past decade, however, that object-oriented programming languages have come into their own. This was due, in a large part, to the development of the C++ programming language in the early 1980's [3]. Object-oriented programming has become the premier programming method of the 1990's, solving many of the challenges of software engineering this decade. This popularity is, in part, a result of the object-oriented programming language's ability to model the essence of a real world problem without the need to model all its non-essential details.

Instead of creating a new data model from scratch, the database researcher has adapted some features and constructs of the existing object-oriented programming languages to create an object-oriented data model. The object-oriented approach allows a more direct modeling of real-world database applications, a more natural view of the application to the user. The object-oriented data model encapsulates the structure of objects with their permitted operations while, at the same time, hiding the details of how the structure and operations are implemented in the database system [4]. Additionally, this approach allows objects to be shared by multiple applications and reduces both the code and storage requirements for the database. Complex objects can be represented directly by the object-oriented data model without having to distort into tuples, as in the relational model. These object-oriented constructs provide the features we desire for our new database applications, allowing the easy construction and maintenance of complex (application) databases.

As an example of an object-oriented approach for Computer-Aided Design, let us look at the application of designing computer chips. The basic design of a chip, although complex, has been established and does not change. This structure of a chip may be described as a class in object-oriented terminology. There are then additional chips which perform specific functions, such as floating-point operations, and are variations of the basic chip design. Again, in object-oriented terminology this may be thought of as a *specialization* of the basic chip design. These different chips, i.e., object classes, form a hierarchy where one class inherits the design from another, again a feature of object-oriented database construct. Finally, the varied ways in which the chips may be interconnected can be modeled by another object-oriented concept, i.e., the covering. As it can be seen, the object-oriented features provide the necessary structures and constructs for this type of application.

From the above paragraphs, we learn that the O-ODM is needed to support the new and complex database applications. This data model provides the user the means to specify, create, and interact with an object-oriented database for a new complex application. To be able to specify for the application a new data definition language, i.e., the new object-oriented data definition language or O-ODDL, is required. The design and specification of the O-ODDL to support this new O-ODM is the thrust of my thesis.

This thesis consist of five chapters. Chapter I consist of this introduction. In Chapter II, I introduce and explain the object-oriented features and constructs which make up the new data model. Once the features and constructs have been established, I develope the actual specification for the O-ODDL in Chapter III. In Chapter IV, I utilize the newly designed O-ODDL and specify an example database application. In Chapter V, I summarize the result of my thesis along with some follow-on research topics.

II. OBJECT-ORIENTED FEATURES AND CONSTRUCTS

In this chapter, I define and examine the features and constructs that form the basis of the object-oriented data model. Unlike the traditional database models mentioned in Chapter I, there is no agreed specification defining an object-oriented data model. I borrow those features and constructs which define object-oriented programming languages and incorporate them into the object-oriented data model. The features and constructs [1, 5] of this object-oriented data model are as follows:

- Every entity is represented by an object. Each object is assigned a unique identifier.
- An object is represented by a set of attributes and methods. This set represents the structure and behavior of an object.
- Classes consist of like objects, i.e., objects having common attributes and methods.
- Inheritance. Relationships among object classes in a hierarchy.
- Covering. Relationships among object classes in different hierarchies.

A. ATTRIBUTES AND METHODS

An *attribute* is a variable which is defined by a name and a type. The *type* of an attribute may be either *simple* or *complex*. A simple attribute is atomic, or not divisible, such as an integer or character. A complex attribute consist of a set of attributes and can be divided into smaller parts, such as another class. The values of the attribute are limited to their designated *domain* and, in turn, are defined by the attribute type. An attribute is defined for either a single object (i.e., singleton) or a set of objects. A singleton forms a one-to-one (1:1) relationship from the attribute to the object whereas a set forms a one-to-many (1:N) relationship from attributes to objects of the set.

A *method* may be defined as an operation that manipulates an object. The defined methods for a class provides the user the means to interface with an object of the class. A method consist of two parts. The first is the *signature*. The signature contains the name of the method, the name and type of parameters, and the return type. The second part of the

method is the *implementation*. The implementation is the code which executes the method. This code is written in a programming language and is not visible to the user. [6]

B. OBJECTS AND OBJECT CLASSES

An object is the most basic and fundamental construct in the object-oriented data model. Objects are collections of specific attributes and the methods allowed on them. An object must be persistent, i.e., permanently stored in the object-oriented database, when used in a database application. However, in an object-oriented programming language, an object exists only during the program execution.

There are two types of objects, *simple* and *complex*. A simple object consist of only a basic data type such as a character, a character string, an integer, or a number. These simple objects are sometimes referred to as *primitive objects* [1]. These primitive objects are provided by the object-oriented database system. The second type of objects are complex, or composite. A complex object simply consist of more than one object, simple or complex.

A *class* is created by grouping together objects, all of which share the same attributes and methods. A class serves as the template, or schema, from which an instance, i.e., an object, may be identified. However, the class contains no data except its instances. As an example, consider characters. Each character is a primitive object of character type which is the name of the class itself. A class is therefore defined by two parts, a set of attributes and a set of methods. The set of attributes define the structure of the data to be stored in the database for each instance of the class, or in other words for each object. The set of methods define the operations permitted on an object of the class. The definition of a class forms an abstract data and operational type. An abstract data or operational type hides the details of their implementation from the user and allows the user to access either the attribute values or methods by their names. A set of classes define the schema for the object-oriented database.

Encapsulation, a feature used extensively in object-oriented programming languages, is related to an abstract data and operational type. It is important in that it provides a means to separate the specification of an object from its implementation [6]. Although not implemented in the traditional database models, the feature of encapsulation is now part of the object-oriented data model.

An object consist of a data part and an interface part. The data part is the data structure along with the implementation of the methods. The interface part consist of the name and arguments of each method. The object is accessible only through this interface. The user of an object is aware only of the interface for the object. The internal details are hidden from the user via the encapsulation which provides the ability to modify the internal structure and/or the implementation details of the methods without affecting the external applications which reference the objects.

In object-oriented programming languages, an object is totally encapsulated. *Totally encapsulated* means that all operations are defined as part of the class and unavailable to other classes. This total encapsulation would be too restrictive in the object-oriented data model. Flexibility is provided through the object-oriented data definition language to allow the user to construct queries which may apply to all the classes.

C. OBJECT IDENTIFIERS

Objects in the object-oriented data model are unique and distinguishable from each other, since each object is assigned a system-defined *object identifier* or *OID*. The OID represents a unique object. The OID performs an important function in the object-oriented data model, allowing the sharing of objects. This sharing of objects accomplishes two things. First, data sharing reduces the physical storage requirements of the database, since shared objects are not duplicated in storage. Secondly, data sharing simplifies the update problem, requiring one update on an object which may be shared by many objects. Data sharing therefore contributes in maintaining the integrity of the database.

Consider the following: An object of the Student class consist of a name, major

and a description of a course the student is taking. Another object of the Faculty class consist of a name, rank, and a description of a course the faculty member is teaching. These objects may be represented as follows:

{student_x, CS, (CS4001, databaseI, S429)}
{student_y, CS, (CS4001, databaseI, S429)}
{faculty_z, professor, (CS4001, databaseI, S429)}

The information on the course is duplicated for each object. Let another object of the Course class be defined consisting of the course number, course name, and room number with a unique OID. Now the same information as before may be represented as:

{CS4001, databaseI, S429}
{student_x, CS, (OID_course)}
{student_y, CS, (OID_course)}
{faculty_z, professor, (OID_course)}

Any modification to the course object will be reflected in objects for the student and faculty. A modification of the course now requires one update of an object versus many updates of three objects. The physical storage requirements are also reduced, since OIDs are shorter than course information. This is critical in applications such as multimedia where objects may occupy pages versus bytes of data.

D. INHERITANCE

Inheritance establishes relationships between two or more classes. These class relationships are critical to our data model. Inheritance as defined in [1] has the following definition:

Class B is defined as a *subclass* of class A if class B inherits all the attributes and methods of class A. In this definition, class B contains all the attributes and methods of class A. Class A, however, need not contain all the attributes and methods of class B. If otherwise, A and B should be the same class. The difference is class B contains additional attributes and/or methods in addition to those defined for class A. This inheritance relation forms a hierarchy of classes A and B.

Because class B inherits all the attributes and methods of class A, class B can be thought of as *inheriting* class A. Class B contains additional attributes and/or methods not contained in class A. Thus class B is a *specialization* of class A. Conversely, class A may be considered a *generalization* of class B.

Two types of inheritance results from this hierarchical relationship among classes, data and operational inheritance [1]. *Data inheritance* results in the strong typing of objects in the hierarchial relationship. *Strong typing* results from the objects in the hierarchy being created from the same set of attributes. *Operational inheritance* results from the sharing of the same methods by all objects in the hierarchy. The combination of data and operational inheritance contributes to the *integrity* of the database, a must in any database design.

The following example illustrates the inheritance feature of the object-oriented data model for a University database containing both the Student and the Faculty classes:

Each Student is defined by a name, address, major, and a set of courses taking. Methods applied to a Student include add_course and change_address.

Class: Student
Attributes: name
address
major
courses_taking
Methods: add_course
change_address

Each Faculty is defined by a name, address, department, and a set of courses teaching. Methods applied to a Faculty include assign_course and change_address.

Class: Faculty
Attributes: name
address
department
courses_teaching
Methods: assign_course
change_address

Both Student and Faculty share common attributes (i.e., name and address) and the same method (change_address). Using the inheritance principle, we are able to group the shared attributes and methods into a common superclass named Person. Person will be defined by the common attributes (name and address) and the common method (change_address). Both Student and Faculty then become a specialization or subclass of Person. Specialization refers to inheriting all of Person's attributes and methods in addition to specific attributes and/or methods of its own. Student is said to inherit Person with two additional attributes (courses_taking and major) and an additional method (add_course). Faculty also inherits Person with two additional attributes (courses_teaching and department) and an additional method (assign_course). Figure 1 demonstrates the inheritance property graphically.

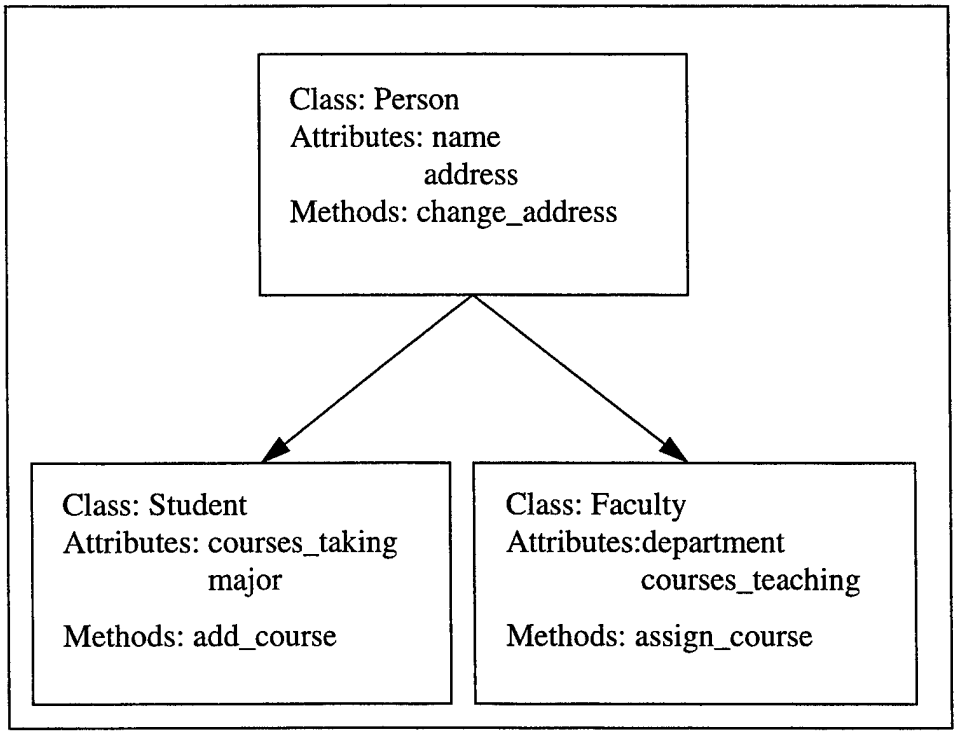


Figure 1. Inheritance Hierarchy of Three Classes

As a result of the inheritance only one definition of the common attributes and shared methods is written and maintained, thereby maintaining the database integrity.

E. COVERING

Covering is a relationship defined on objects of different classes in the object-oriented data model. The following definition of covering is taken from [1]. Two classes are said to have a *covering* relationship if every object of one class, A, maps to, or corresponds to, a subset of objects, b's, from a second class, B. In this instance class A is said to cover class B. Class A is referred to as the *covering class* and class B is referred to as the *member class*. Covering may further be defined mathematically. All the subsets of objects, b's, form the power set of class B or P(B). There exist a mapping function or correspondence, f, which determines for an object, a, from class A all the objects, b's, of the subset f(a) from class B such that f(a) = b. This can be expressed as f: A -> P(B).

As an example of covering consider the following example consisting of the following two classes, Project and Student:

Class: Project
Attributes: project_name
advisor

Class: Student
Attributes: student_name
student_address
major

A student may be part of a project, such as the student's thesis. The student may be working on the project individually or as a member of a group or team. Each project in turn has a project name and an advisor. Here the project_name is the correspondence f that maps an object from Project to a subset of the set of objects from Student, i.e., f: Project -> P(Student). A project may map to the set of students who are members of that project, if several students are working on various theses in the same project. In Figure 2, I demonstrate the covering feature.

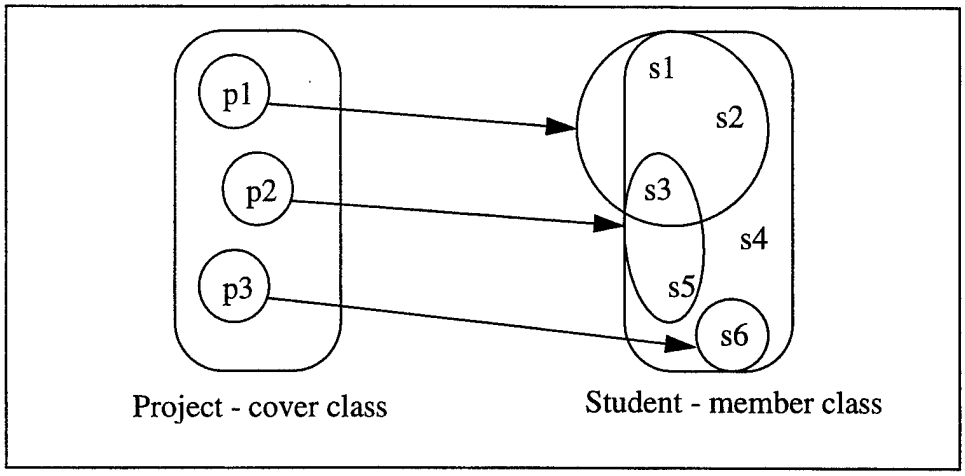


Figure 2. The Covering Relationship

As illustrated in Figure 2 the covering does not have to partition the member class. Specifically, s4 is not in any covering and s3 participates in two coverings. An object in the member class is not restricted in the number of covering relationships it may participate in. The number of relationships may be one, many, or none. We note that s1, s2, and s5 participate in one, s3 in two and s4 in none.

Covering therefore forms a relationship of an object in one class and a subset of objects of another class, instead of directly forming an inheritance hierarchy of classes. Covering permits an object of the cover class to be operated on either as a set of member classes or as a singleton (member class).

III. THE DATA-DEFINITION-LANGUAGE SPECIFICATION

The features and constructs described in Chapter II are the basis for the object-oriented data model. The object-oriented-data-modelled database is specified, in turn, in an object-oriented data definition language (O-ODDL). The O-ODDL provides the syntax necessary to specify the object-oriented-data-modelled database.

Prior to an understanding of the O-ODDL, it is necessary for the reader to have a basic understanding of the system on which it is intended to be implemented. This understanding is essential to the design of the O-ODDL.

A. THE BACKGROUND

The system used in the Laboratory for Database Research at the Naval Postgraduate School is the Multimodel and Multilingual Database System (M^2DBS). See Figure 3 for an organization of its databases and model/languages interfaces. It is not my intention to give a complete description of M^2DBS here, but only to describe its effect on the design of the O-ODDL. A more complete explanation of M^2DBS is contained in [7]. The M^2DBS is designed to support heterogeneous databases of various models. Each database is stored in the system in the format of the kernel data model in lieu of its own data model. Currently M^2DBS supports databases in the relational, hierarchial, network, and functional data models. In order to support a database in the new data model, the O-ODM, it is necessary to provide an object-oriented-data-model interface for M^2DBS.

The kernel data model for the M^2DBS is the attribute based data model (ABDM). Because the ABDM serves as the kernel, our new database in the O-ODM must be mapped to its ABDM equivalent. It is this mapping to the ABDM equivalent that impacted the design of the specification of the O-ODDL. The impact on the design is discussed in the following sections of this chapter.

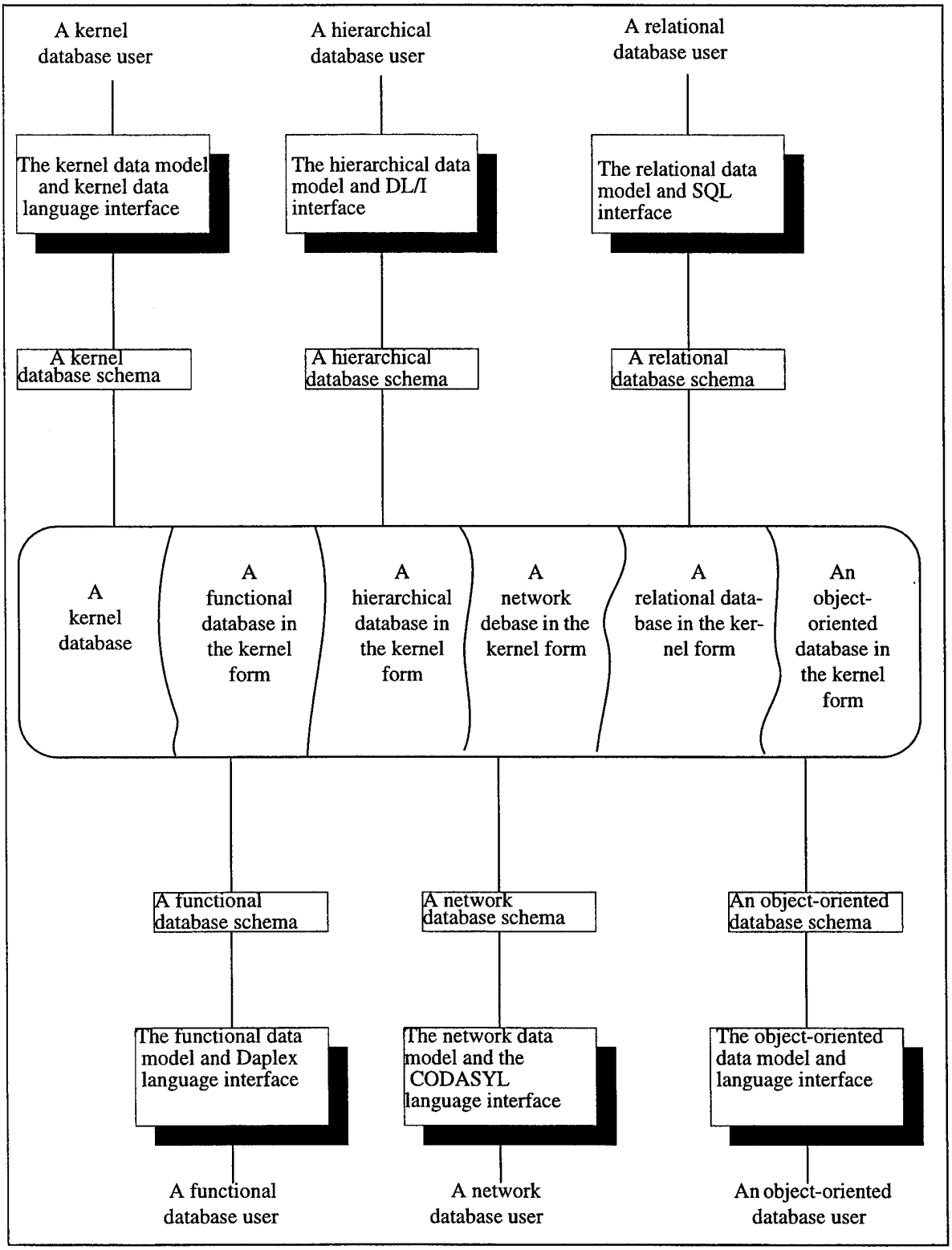


Figure 3. The Multimodel and Multilingual Database System.

The ABDM's basic construct is the attribute-value pair. An attribute-value pair is represented as <attribute, value>. Attribute, in this instance, represents the domain for the corresponding value. A record consist of a set of attribute value pairs such that: (1) no attribute is repeated, (2) an attribute can only have one value in a record, and (3) every record must have at least one key.[7]

B. THE DATA DEFINITION LANGUAGE

Now that the M^2DBS and attribute-based data model have been mentioned, I present the specification for the O-ODDL. From this specification we are able to create a schema for an object-oriented database. The schema is a template describing the attributes and the relations in the database. From Chapter II, the relations which must be specified are the class, inheritance, covering, attributes, and value types representing a set. A description of these specifications is contained in the following subsections. The conventions used in defining the specifications are:

- reserved words in bold-faced,
- class names begin with an uppercase letter followed by lower-case letters,
- attribute names and types in lower case,
- words in italics being provided by the user,
- attributes being either simple, i.e., single-valued or composite, i.e., a set of values.

1. The Class Specification

From Chapter II, the definition of a class is the grouping of objects which share common attributes and methods. Therefore, the specification of a class must contain the ability to specify these attributes and methods. Figure 4 contains the specification of a class for our O-ODM.

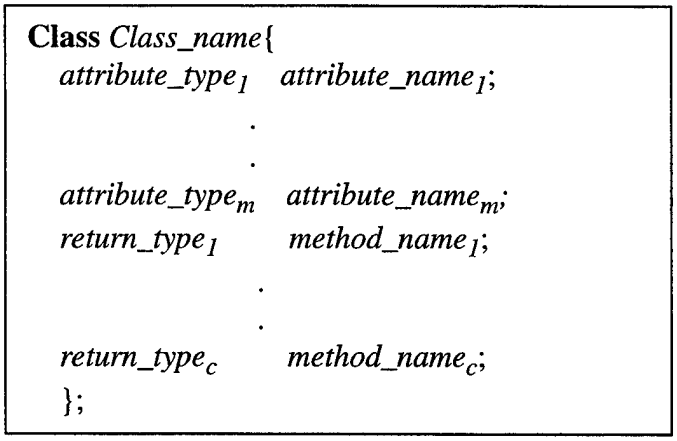


Figure 4. Specification of a Class.

Here *attribute_type,* represents the domain of the attribute. The domain may be of a simple type, such as an integer or character, or a complex type, such as another class. The *attribute_name* is the name assigned to the attribute and is the placeholder for its value. This value may represent either a singleton or a set.

2. The Inheritance Specification

Inheritance is the second feature of our O-ODM for which a specification is to be provide. From Chapter II, we learn that the inheritance is described as a hierarchy. In this hierarchy, a class assumes, or inherits, all the attributes and methods from another class. Figure 5 is the specification for the inheritance feature of our O-ODM.

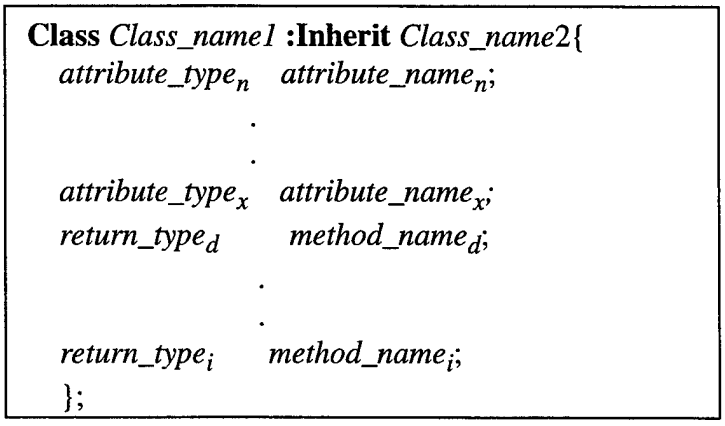


Figure 5. Specification for Inheritance.

Here, *Class_name1* refers to the subclass, or specialization, of *Class_name2* which is the superclass, or generalization. The attributes and methods contained in this specification are those which are unique to *Class_name1*, but not part of *Class_name2*.

3. The Cover Specification

The third specification to be defined is that for the covering. From Chapter II, we recall that the covering defines a mapping, or relationship, between an object in a class, the cover class, to a set of objects in a second class, the member class. This covering specification is provided in Figure 6.

Class *Class_name1* **:Cover** *Class_name2*{
attribute_type$_1$ *attribute_name*$_1$;
.
.
attribute_type$_m$ *attribute_name*$_m$;
return_type$_1$ *method_name*$_1$;
.
.
return_type$_c$ *method_name*$_c$;
};

Figure 6. Specification for the Covering

In this instance *Class_name1* represents the cover class and *Class_name2* represents the member class. This means that an object of class *Class_name1* maps to a subset of objects from class *Class_name2*.

4. Set Relationships

First in order for an attribute to assume the values of a set, a scheme is devised. Recall that in the ABDM, an attribute-value pair can only contain a singleton value, not a set. Therefore, a way must be devised to represent this set. I elect to create a new attribute_type to represent a set relation. This new attribute_type, called *set_of*, is implemented in the kernel. The kernel creates a template in ABDM format containing the OID's of the objects in the participating classes. An example is the best way to illustrate this feature. This creation also shows that the introduction of O-ODM constructs require some

modifications and enhancements of ABDM constructs.

Consider a simplified definition of the class Student. In this example, Student consist of two attributes, name and schedule. Name is a character string and schedule refers to the set of courses of class Course which a student is taking.

```
Class Student{
  char*   name;
  set_of Course   schedule;
  };
```

The user would logically expect the record to appear similar to the following:

{<TEMP, Student>, <name, John Doe>, <schedule, {c1, c2, c3}>}

As mentioned previously, the ABDM cannot store a set such as {c1, c2, c3}. In this case, the values of the set will be stored in a separate table containing the OID for the Student object and the OID for the Course object. The new template for this table would look like:

{<TEMP, Student_course>, <char*, OID_Student>, <char*, OID_Course>}

Each record contains only one student and one course. For the above example three records are created. Each record with the same student OID, but different course OID's. To retrieve the student's schedule the table would be searched with the entering argument of the students OID. None of this is visible to the user however. Entries in the data dictionary directs the system to the correct number of course records.

The second attribute_type created for a set relationship is *inverse_of*. Inverse_of is the complement of set_of. In this case, two classes each have a set relationship referring to the other. If only set_of is used, two identical tables will be generated by the system. Using inverse_of, results in only one table to be generated. In this way we prevent duplicate tables, save the storage space, and maintain the integrity of the database.

In addition to class Student, there is a second class Course. Course consists of two attributes: course_name and roster. Course_name is a character string and roster refers to the set of students, of the class Student, who are taking the course.

```
Class   Course{
   char* course_name;
   inverse_of Student.schedule    roster;
   };
```

Here, roster is the inverse set relationship of a student's schedule (i.e., Student.schedule). In other words, for every course that a student is taking the course is in the student's schedule; conversely, the student should be on the course's roster. Schedule refers to a set of objects in Course, while roster refers to a (an inverse) set of objects of class Student.

The specifications for set_of and inverse_of appears in Figure 7.

set_of *Class_name attribute_name*;

inverse_of *Class_name.attribute_name attribute_name*;

Figure 7. Specification for Set_of and Inverse_of.

This specification of the set_of and inverse_of constructs allows this O-ODM to represent 1:N and M:N relationships of two object classes.

IV. THE FACULTY-STUDENT DATABASE

In order to illustrate the O-ODDL provided in Chapter III, I create an object-oriented database from a real world application. This database is named the Faculty-Student database and represents a database found at a University. For this application of the O-ODDL, only attributes will be considered, i.e., no methods. First a description of the database to be constructed is given. Secondly, from the description of the database, the conceptual database design is accomplished. Finally the conceptual database design is translated to its object-oriented specification in the O-ODDL. This translation, or mapping, corresponds to the logical database design.

A. THE DATABASE DESCRIPTION

Prior to specifying the conceptual database design I first establish the requirements of the Faculty-Student database. These requirements define the objects to be represented and the relationships between objects. The Faculty-Student database contains faculty members, students, courses, and teams (i.e., projects) described as:

- Either a faculty member or a student is defined by a name (fname, mi, lname), an address (street, city, state and zipcode), and a sex.
- A faculty member is further defined by a department (dept), a rank or a title (i.e., military or civilian faculty), the teams he or she advises, and the courses he or she teaches.
- Students are further defined by the courses the student takes (student's schedule), a major, and a student number (student_no).
- A course is defined by a course name (cname), a course number (cse_no), a section number (sec_no), an instructor, and a set of students enrolled in the course (course's roster).
- A team is defined by a project name (prjname) and the faculty advisors.

Based on this description, the Faculty-Student database initially consist of four classes. The four classes are Faculty, Student, Course, and Team. The constraints and rela-

tionships imposed on the four classes of the Faculty-Student database are now defined as:

- Every course can have one and only one instructor and at least one student.
- A faculty member may teach more than one course.
- Every student is enrolled in at least one course.
- Every team has an advisor and at least one student.
- Faculty members are of two types, either military or civilian. However, only the civilian faculty member may be the advisor of a team.

B. THE CONCEPTUAL DATABASE DESIGN

The goal of the conceptual database design phase is to produce a conceptual schema for the Faculty-Student database using a high level (i.e., close to the user's view) data model. The conceptual schema is a concise description of the user's requirements (i.e, database description) capturing the aspects of the real world similar to the users perception. To meet this end the conceptual schema must be simple to understand, have a small number of basic concepts, and represent the database in a diagrammatic manner [8]. There are a number of approaches in developing the conceptual database design, i.e., the entity relationship (ER) approach, the enhanced ER (EER) approach and the object-oriented approach. I choose the object-oriented approach as this is the data model of the database.

In the conceptual schema, Figure 8, I identify the classes, the attributes, and the relationships. Applying the object-oriented features and constructs introduced in Chapter II, I represent the Faculty-Student database in object-oriented terms. The attributes may be either simple or complex. The relationships include inheritance, the covering, a singleton, i.e., 1:1, or a set, i.e., 1:N or M:N.

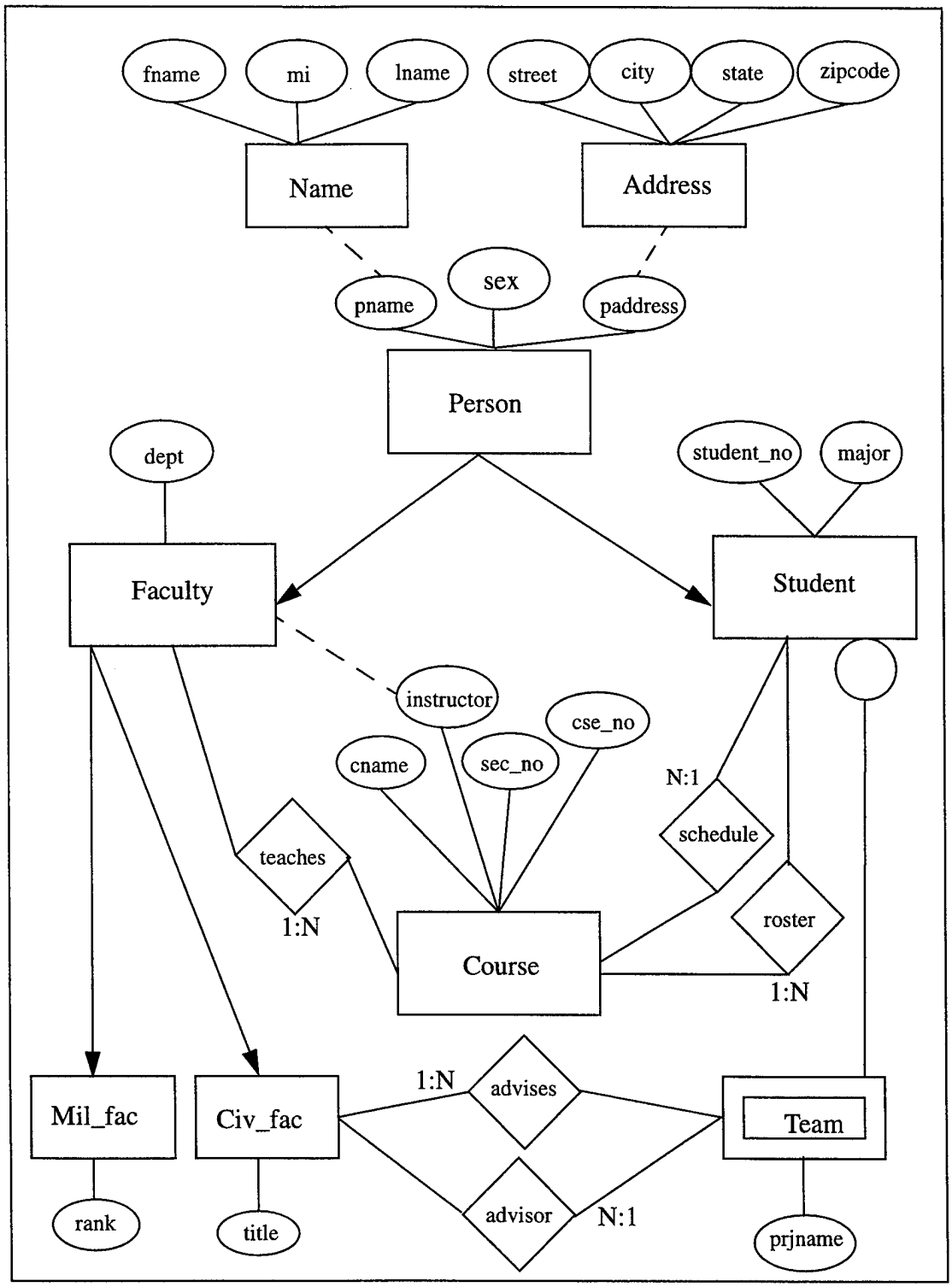


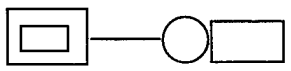
Figure 8. Conceptual View of the Faculty-Student Database

Before continuing further I define the symbols and notations used to represent the conceptual schema in diagrammatic form.


Represents a class.


Represents a singleton type attribute, i.e., 1:1.


Represents a set type attribute, i.e., 1:N.


The covering. The double box represents the cover class and the single box represents the member class.


Inheritance. The arrow points from the superclass to the subclass

Links an attribute whose attribute type is another class to that class

Since both the Faculty and the Student classes contain name, address, and sex, their common attributes are combined into a single superclass named Person. Thus, the Faculty and Student classes inherit these attributes from the Person class. Two types of Faculty are also defined, civilian (Civ_fac) and military (Mil_fac). The Faculty class now becomes the superclass for the Civ_fac and the Mil_fac classes, containing the common attributes dept and teaches. I have created two levels of inheritance in this example, Person -> Faculty -> Civ_fac or Mil_fac, as depicted in Figure 8.

Within the superclass Person, the attributes for fname, mi, and lname may be defined as a single attribute, pname, of attribute type class Name. The attribute type, i.e., Name, in this case is another class. Similarly, the individual attributes comprising a person's address can be combined into a single attribute, paddress, of attribute type class Address. See Figure 8 for the Name and Address classes.

The relationship between Team and the students making up a team can be represented using the covering. Here, Team is the cover class and Student is the member class. From the definition of the covering in Chapter II, this means that an object of class Team maps to a subset of objects of the class Student. This mapping says that a team may have

one or more students as team members. This is depicted in Figure 8.

C. LOGICAL DATABASE DESIGN

The purpose of the logical database design is to create the database in the data model of the chosen DBMS [8]. The high level data model will be mapped to the data model of the implementation. The result of the mapping is the DDL statements in the O-ODM. In this case, I take the conceptual schema, Figure 8, and specify it in the O-ODDL. The description then becomes a specification of classes, inheritances, and covering in the O-ODDL.

For detailed specifications of individual classes and their built in relationships, I translated the conceptual database schema to a schema in the O-ODDL, depicted in Figure 9. In class Faculty, teaches represents an attribute defined by a set, i.e., 1:N, and its type is therefore set_of. Another set_of attribute type is found in Student, the schedule attribute. A third set_of type attribute is found in class Course, the roster attribute. Recall from Chapter II that roster has an inverse relationship with the schedule attribute in class Student. In other words, the set of students (i.e., objects) of class Student, which take the same course, (i.e., have as a member of the attribute schedule of a unique object) of class Course, is identical to the roster for the course (i.e., for an object of class Course). Roster's attribute type therefore is inverse_of Student.schedule. Class Team contains the final set_of type attribute, the advisor attribute. Advisor further has an inverse relationship with the advises attribute in class Civ_fac. Advises attribute type is therefore the inverse_of Team.advisor.

I have now completed the logical database design. The result of this design, Figure 9, is a complete object-oriented specification of the Faculty-Student database in the O-ODDL. This object-oriented schema in the O-ODDL is now ready to be compiled and the physical database created.

```
Class Name{
 char*  fname;
 char*  mi;
 char*  lname;
 }

Class Address{
 char*  street;
 char*  city;
 char*  state;
 char*  zipcode;
 }

Class Person{
 char*    sex;
 Name     pname;
 Address  paddress;
 }

Class Faculty :Inherit Person{
 char*    dept;
 set_of Course  teaches;
 }

Class Student :Inherit Person{
 char*  student_no;
 char*  major;
 set_of Course  schedule;
 }

Class Mil_fac :Inherit Faculty{
 char*    rank;
 }

Class Civ_fac :Inherit Faculty{
 char*  title;
 inverse_of Team.advisor  advises;
 }

Class Course{
 char*    cname;
 char*    cse_num;
 char*    sec_no;
 Faculty  instructor;
 inverse_of Student.schedule  roster;
 }

Class Team :Cover Student{
 char*     prjname;
 set_of Civ_fac  advisor;
 }
```

Figure 9. Faculty-Student Database Schema

V. CONCLUSION

In this thesis I provided the specification for an object-oriented data definition language. This object-oriented data definition language permits the specification of an object-oriented database utilizing the object-oriented data model. The object-oriented data model, in turn, borrowed many of the features and constructs of object-oriented programming languages. These features and constructs include the concept of a unique object, object classes, inheritance, encapsulation and the covering. Because of these features, this object-oriented data model is capable of efficiently representing and managing the complex database applications of today. This object-oriented data model further supports conversion to the attribute-based format, the kernel data model of M^2DBS in the Laboratory for Database Research at the Naval Postgraduate School. This conversion will now allow the development of an object-oriented interface to the M^2DBS.

A. WORK IN PROGRESS

The specification of the object-oriented data definition language is the first step of a larger project involving a total of eleven thesis students and seven Master's Theses. The project's goal is the development of the Object-Oriented Interface for the M^2DBS. Current work on the object-oriented interface involves incorporating the object-oriented data model and the corresponding data languages as model/language compilers. The data languages for this object-oriented interface include both a data definition language and a data manipulation language. A generalized diagram of the object-oriented interface is provided as Figure 10.

This thesis provides the design criteria and the specification of the O-ODM and O-ODDL. To facilitate writing object-oriented transactions, or queries, an object-oriented data manipulation language (O-ODML) is required. The design and specification of the O-ODML is contained in [9].

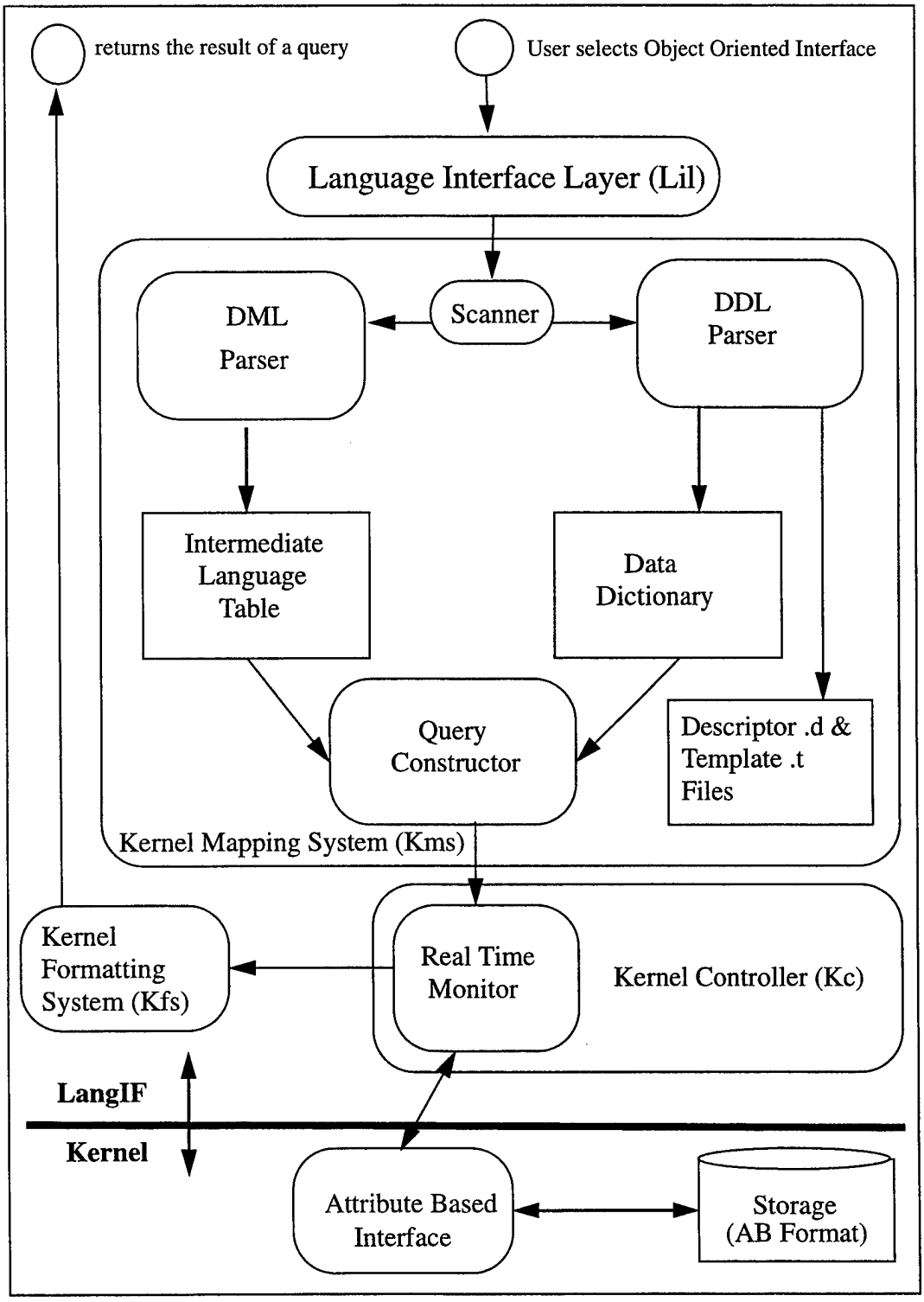


Figure 10. Data Flow of the Object-Oriented Interface

Two compilers are required to support the O-ODDL (O-ODDL Compiler) [10] and O-ODML (O-ODML Compiler) [11]. The O-ODDL Compiler takes as input an object-oriented specification in the form of the O-ODDL. Using this input the O-ODDL Compiler outputs the equivalent ABDM specification and the Data Dictionary. The O-ODML Compiler takes as input an O-ODML statement and produces the equivalent attribute-based data manipulation language (ABDML) statements. Multiple ABDML statements are produced for a single O-ODML statement.

The Real-Time Monitor [12] supervises the execution of the ABDML statements created by the O-ODML compiler. These ABDML statements are executed individually and the final result returned to the user via a Kernel Formatting System.

The final two theses [13, 14] fine-tune the five primary ABDM operations (INSERT, UPDATE, DELETE, RETRIEVE, and RETREIVE-COMMON). These basic operations permit the creation and manipulation of the object-oriented database as an attribute-based database.

B. FUTURE RESEARCH

In the design stage of developing the object-oriented interface for M^2DBS it was decided not to implement methods. The first phase of development was directed towards establishing a working data definition language and data manipulation language. In keeping with the object-oriented paradigm it is recommended future thesis students implement methods. This implementation will provide the user a true external interface, encapsulating the objects. The result is a true object-oriented database.

LIST OF REFERENCES

[1] Hsiao, David K., "The Object Oriented Database Management: A Tutorial On It's Fundamentals", *Proceedings of the Second Far-East Workshop on Future Database Systems*, Kyoto, Japan, April 1992.

[2] Pohl, I., *Object Oriented Programming Using C++*, Benjamin/Cummings Publishing Company, Inc, 1993.

[3] Atkinson, M., et al., "The Object-Oriented Database System Manifesto", *Proceedings of the First International on Deductive and Object-Oriented Databases*, 1989.

[4] Khoshafian, S., *Object Oriented Databases*, John Wiley and Sons, Inc., 1993.

[5] Kim, W., "Object-Oriented Databases: Definition and Research Directions", *IEEE Transactions on Knowledge and Data Engineering*, September 1990.

[6] Bertino, E. and Martino, L., "Object-Oriented Database Management Systems: Concepts and Issues", *IEEE Transactions on Knowledge and Data Engineering*, April 1991.

[7] Hsiao, David K., "Interoperable and Multidatabase Solutions for Heterogeneous Databases and Transactions", a speech delivered at **ACM CSC '95**, Nashville, Tennessee, March 1995.

[8] Elmassi and Navathe, *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company, Inc., 1990.

[9] Stevens, M., *The Design and Specification of an Object-Oriented Data Manipulation Language (O-ODML)*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[10] Ramirez, L. and Tan, R., M., *The Design and Implementation of a Compiler for the Object-Oriented Data Definition Language (O-ODDL Compiler)*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[11] Barbosa, C. and Kutlusan, A., *The Design and Implementation of a Compiler for the Object-Oriented Data Manipulation Language (O-ODML Compiler)*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[12] Senocak, E., *The Design and Implementation of a Real-Time Monitor for the Execution of Compiled Object-Oriented Transactions (O-ODDL and O-ODML Monitor)*, Master's Thesis, Naval Postgraduate School, Monterey, California, September

1995.

[13] Kellett, D. and Kwon, T., *The Instrumentation of a Kernel DBMS for the support of a Database in the O-ODDL Specification*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[14] Clark, R. and Yildirum, N., *The Instrumentation of a Kernel DBMS for the Execution of Kernel Transactions Equivalent to their O-O Transactions*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library 2
   Code 52
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Chairman, Code CS 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

4. Dr David K. Hsiao, Code CS/HS 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

5. Dr C. Thomas Wu, Code CS/KA 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

6. CDR Robert B. Badgett 2
   3229 Brook Highland Trace
   Birmingham, AL 35242